

Web Viewer Client Functions (API)

This section applies to the COM interface of Entire Screen Builder's Web Viewer.

This section assumes that you are familiar with the use of ActiveX controls in applications. You can use ActiveX controls from a wide variety of languages: C++, JavaScript, Visual Basic, Visual Basic for Applications, etc.

The interface consists of a set of properties, methods and events. Properties are needed for the control to know how to make the connection and its visual appearance on screen. The methods are used to handle connection/disconnection and manipulate screen data. The events are used to notify the container about changes in the control's state (e.g. an event is fired when a new screen arrives).

This chapter covers the following topics:

- Screen Interface Methods
- Menu Interface Methods
- Events

See also: *Common Viewer Functions (API)* and *Overview of Client Control Properties*

Screen Interface Methods

The following methods are available:

- AboutBox
- ComposeId
- Connect
- Disconnect
- EnableControl
- GetActiveField
- GetFieldContent
- GetFieldCount
- GetFieldIDFromIndex
- GetTitleBarText
- IsBS2000ToolbarVisible
- IsConnected
- IsStatusbarVisible
- IsTitlebarVisible
- IsToolbarVisible
- PrintCurrentScreen
- PrintDirectCurrentScreen
- RunJavascript
- SendKey
- SendKeyID
- SendKeyIDWithText
- SendKeyWithText
- SetFieldContent
- SetReceiveTimeout

- SetUserNameAndPassword
- Show3270Toolbar
- ShowBS2000Toolbar
- ShowControl
- ShowNextScreens
- ShowProperties
- ShowStatusBar
- ShowTitleBar
- ShowToolbar

AboutBox

This method shows the control's About dialog box.

This dialog box shows the version number of the control and the copyright notice.

Format

```
void AboutBox()
```

Input Parameters

No input parameters.

Return Values

No return values.

ComposeId

To find out the IDs of the automatically generated fields, you can use this method. For example, if you know that there is an input field in the current screen in coordinates "12,7", you can call ComposeId(12, 7) to get the dialog ID of the edit control associated with this field.

Returns IDs for dialog controls when using basic rules. When basic rules are applied, the screen fields are directly translated to dialog controls: output fields in the screen are mapped to static Windows controls, and input fields in the screen are mapped to edit dialog controls.

When using extended dialogs, there is no need to call this method, as the user already knows the IDs of the controls. These IDs are set by the user from the resource editor used to generate the dialog.

Format

```
long ComposeId(long iRow, long iCol)
```

Input Parameters

long iRow Row number.

long iCol Column number.

Return Values

long The Windows control identification.

Connect

This method connects the control to the server.

This method gets its parameters from the control properties and connects to the server with the appropriate method. If an error occurs when connecting, the ActiveX shows it to the user.

After this method has been called, two exclusive events can be fired: `Connected` or `ConnectFailed`. The first is fired if the control succeeds in connecting, the latter is fired if the connection fails.

Format

```
void Connect ()
```

Input Parameters

No input parameters.

Return Values

No return values.

Disconnect

This method forces the control to disconnect.

If an error occurs, the ActiveX notifies the user with a message box. If the control has successfully been disconnected, the `Disconnected` event is fired from the control.

Format

```
void Disconnect ()
```

Input Parameters

No input parameters.

Return Values

No return values.

EnableControl

This method enables or disables a control. It allows individual controls on a dialog to be disabled separately from the repository settings.

Format

```
EnableControl(long ControlID, long Enable)
```

Input Parameters

long ControlID The control ID is usually returned from
 GetFieldIDFromIndex.

long Enable Can either be set to non-zero (enable) or zero (disable).

Return Values

Always true.

GetActiveField

This method returns the control identification with the current focus.

Format

```
long GetActiveField()
```

Input Parameters

No input parameters.

Return Values

long	Control identification.
------	-------------------------

GetFieldContent

This method returns the window text of the control with ID `iFieldId` in the currently shown dialog.

This method can be used for extended and basic dialogs. To find the ID of a control in a basic dialog, use the `ComposeId` method.

For example, to get the contents of the field in row 12 and column 7 in a basic dialog, call the following:

```
control.GetFieldContent( control.ComposeId( 12, 7 ) )
```

For example, to get the name of a button with ID 34572 in an extended dialog, call the following:

```
control.GetFieldContent( 34572 )
```

Format

```
BSTR GetFieldContent(long iFieldId)
```

Input Parameters

<code>long iFieldId</code>	Control identification in the dialog.
----------------------------	---------------------------------------

Return Values

<code>BSTR</code>	A string with the field content.
-------------------	----------------------------------

GetFieldCount

This method returns the number of all fields in the current screen.

You can iterate all controls as follows:

```
int iTotal = m_cNWWClient.GetFieldCount();
for (int iLoop = 0; iLoop < iTotal; iLoop++)
{
    int iTestItem = m_cNWWClient.GetFieldIDFromIndex(iLoop);

    // Now iTestItem can be used in functions needing
    // a control ID

}/* End for : iLoop of available fields */
```

Format

GetFieldCount

Input Parameters

No input parameters.

Return Values

Total of fields.

GetFieldIDFromIndex

This method returns the field ID of the *n*th field of the screen. See the GetFieldCount sample.

Format

```
GetFieldIDFromIndex(long FieldIndex)
```

Input Parameters

long FieldIndex	The index of the field. For example, the 10th field on the screen.
-----------------	--

Return Values

Field ID or 0 if not found.

GetTitleBarText

This method returns the text that is defined for the title bar.

Format

```
GetTitleBarText
```

Input Parameters

No input parameters.

Return Values

String containing the current title.

IsBS2000ToolbarVisible

This method returns true if the BS2000 toolbar is currently visible.

Format

```
BOOL IsBS2000ToolbarVisible()
```

Input Parameters

No input parameters.

Return Values

- | | |
|-------|------------------------------------|
| true | The BS2000 toolbar is visible. |
| false | The BS2000 toolbar is not visible. |

IsConnected

This method returns true if the control is connected.

Format

```
boolean IsConnected ()
```

Input Parameters

No input parameters.

Return Values

true	The viewer is connected to the server.
false	The viewer is not connected to the server.

IsStatusBarVisible

This method returns true if the status bar is currently visible.

Format

```
BOOL IsStatusBarVisible()
```

Input Parameters

No input parameters.

Return Values

true The status bar is visible.

false The status bar is not visible.

IsTitlebarVisible

This method returns true if the title bar is currently visible.

Format

```
BOOL IsTitlebarVisible()
```

Input Parameters

No input parameters.

Return Values

true The title bar is visible.

false The title bar is not visible.

IsToolbarVisible

This method returns true if the toolbar is currently visible.

Format

```
BOOL IsToolbarVisible()
```

Input Parameters

No input parameters.

Return Values

true The toolbar is visible.

false The toolbar is not visible.

PrintCurrentScreen

This method prints the contents of the current screen on a printer that is defined in Windows. The Print dialog box appears in which you can select another printer and/or modify the print properties.

You can either print the GUI screen on which the transformation rules are applied or the character screen (i.e. the actual screen as sent from the host).

Format

```
BOOL PrintCurrentScreen(long IPrintType);
```

Input Parameters

long IPrintType	Can either be set to 0 (character screen) or 1 (GUI screen).
-----------------	--

Return Values

True if printed.

PrintDirectCurrentScreen

This method immediately prints the contents of the current screen on the default printer that is defined in Windows. The Print dialog box does not appear.

You can either print the GUI screen on which the transformation rules are applied or the character screen (i.e. the actual screen as sent from the host).

Format

```
BOOL PrintDirectCurrentScreen(long IPrintType);
```

Input Parameters

long IPrintType	Can either be set to 0 (character screen) or 1 (GUI screen).
-----------------	--

Return Values

True if printed.

RunJavascript

This method executes any JavaScript procedure on the Entire Screen Builder Server.

The JavaScript routine must be specified in pcJavaScriptProcedure.

Format

```
void RunJavascript(BSTR pcJavaScriptProcedure)
```

Input Parameters

BSTR pcJavaScriptProcedure The JavaScript procedure name.

Return Values

No return values.

SendKey

This method simulates a key stroke for the current dialog. You must specify the ID of the control to which the cursor is to be positioned.

The key name must be a string from the following list:

Key Name	Description
ATTN	Send the "Attention" key to the server.
CLEAR	Send the "Clear" key to the server.
PA1 through PA3	Send the specified application key number to the server.
PF1 through PF48	Send the specified function key number to the server.
RESET	Send the "Reset" key to the server.
RETURN	Send the "Enter" key to the server.
SYSREQ	Send the "System Request" key to the server.

To send PF keys in AS/400 style, the session has to be defined as an AS/400 session. See *Communication Properties for Telnet TN3270* in Entire Screen Builder's *System Management Hub* documentation. Entire Screen Builder provides a key scheme with a special layout for AS400 host systems. See *Key Schemes* in Entire Screen Builder's *System Management Hub* documentation.

If the key name is not found in this list, a "RETURN" will be sent.

Format

```
void SendKey(BSTR szKeyName, long iFieldId)
```

Input Parameters

BSTR szKeyName Key name to be sent to the server.

long iFieldId Field identification of the control to which the cursor is to be positioned.

Return Values

No return values.

SendKeyID

This method simulates a key stroke for the current dialog. You must specify the ID of the control to which the cursor is to be positioned.

The key ID must be one from the following list:

Key Name	Key ID
RETURN	0x7D
SYSREQ	0xF0
CLEAR	0x6D
ATTN	0x7E
RESET	0x03
PA1	0x6C
PA2	0x6E
PA3	0x6B
PF1	0xF1
PF2	0xF2
PF3	0xF3
PF4	0xF4
PF5	0xF5
PF6	0xF6
PF7	0xF7
PF8	0xF8
PF9	0xF9
PF10	0x7A
PF11	0x7B
PF12	0x7C
PF13	0xC1
PF14	0xC2
PF15	0xC3
PF16	0xC4
PF17	0xC5
PF18	0xC6
PF19	0xC7
PF20	0xC8

Key Name	Key ID
PF21	0xC9
PF22	0x4A
PF23	0x4B
PF24	0x4C
PF25	0x4D
PF26	0x4E
PF27	0x4F
PF28	0x50
PF29	0x51
PF30	0x52
PF31	0x53
PF32	0x54
PF33	0x55
PF34	0x56
PF35	0x57
PF36	0x58
PF37	0x59
PF38	0x5A
PF39	0x5B
PF40	0x5C
PF41	0x5D
PF42	0x5E
PF43	0x5F
PF44	0x60
PF45	0x61
PF46	0x62
PF47	0x63
PF48	0x64

If the key ID is not found in this list, a "RETURN" will be sent.

Format

```
void SendKeyID(long KeyID, long iFieldId)
```

Input Parameters

long KeyID Key ID to be sent to the server.

long iFieldId Field identification of the control to which the cursor is to be positioned.

Return Values

No return values.

SendKeyIDWithText

This method simulates a key stroke for the current dialog. It also allows you to send text with the key stroke, for example, a user ID or password. The text cannot be seen by the user.

Format

```
void SendKeyIDWithText(BSTR TextToSend, long FieldId, long KeyID, long KeyFieldId)
```

Input Parameters

BSTR TextToSend	Text to be sent to the host.
long FieldId	Field identification of the control to which the text is to written.
long KeyID	Key ID to be sent to the host. See the <code>SendKeyID</code> method for all valid key IDs.
long KeyFieldId	Field identification of the control to which the cursor is to be positioned.

Return Values

No return values.

SendKeyWithText

This method simulates a key stroke for the current dialog. It also allows you to send text with the key stroke, for example, a user ID or password. The text cannot be seen by the user.

Format

```
SendKeyWithText(BSTR TextToSend, long FieldId, BSTR KeyName, long KeyFieldId)
```

Input Parameters

BSTR TextToSend	Text to be sent to the host.
long FieldId	Field identification of the control to which the text is to written.
BSTR KeyName	Key name to be sent to the host. See the <code>SendKey</code> method for all valid key names.
long KeyFieldId	Field identification of the control to which the cursor is to be positioned.

Return Values

No return values.

SetFieldContent

This method sets the text of a control. It is the complement of GetFieldContent.

The new text of the control must be specified in pcContent.

Format

```
void SetFieldContent(long iFieldId, BSTR pcContent)
```

Input Parameters

long iFieldId	Field identification of the control in which data has to be replaced.
BSTR pcContent	Data.

Return Values

No return values.

SetReceiveTimeout

This method defines how long the ActiveX is to wait for data before a timeout occurs.

This method can be called once the ActiveX is connected. A good place for calling this method is in the `Connected()` event which is fired when the connection with the server has been established.

When the ActiveX has not received any data after the defined number of seconds, the communication with the server is closed and a message box is displayed indicating the error.

Format

```
void SetReceiveTimeout(long ITimeout)
```

Input Parameters

<code>long ITimeout</code>	Number of seconds. When you specify 0, the ActiveX waits infinitely; a timeout does not occur.
----------------------------	--

Return Values

No return values.

SetUserNameAndPassword

This method lets the client connect to the Entire Screen Builder Server using the user name and password passed in the first and second parameter.

This method must be called before the Connect method. When the SetUserNameAndPassword method is called, the User Authentication dialog box is not shown.

Format

```
void SetUserNameAndPassword(BSTR pcUsername, BSTR pcPassword)
```

Input Parameters

BSTR pcUsername	The user name used to connect to the Entire Screen Builder Server.
BSTR pcPassword	The password used to connect to the Entire Screen Builder Server.

Return Values

No return values.

Show3270Toolbar

This method shows or hides the 3270 toolbar.

Format

```
Show3270Toolbar(long Show)
```

Input Parameters

long Show	Can either be set to non-zero (show) or zero (hide).
-----------	--

Return Values

No return values.

ShowBS2000Toolbar

This method shows or hides the BS2000 toolbar.

Format

```
ShowBS2000Toolbar(long Show)
```

Input Parameters

long Show	Can either be set to non-zero (show) or zero (hide).
-----------	--

Return Values

No return values.

ShowControl

This method shows or hides a control.

This is the same as EnableControl. It only hides instead of enabling.

Format

```
ShowControl(long ControlID, long Show)
```

Input Parameters

long ControlID	The control ID is usually returned from GetFieldIDFromIndex.
----------------	---

long Show	Can either be set to non-zero (show) or zero (hide).
-----------	--

Return Values

Always true.

ShowNextScreens

This method shows the next screens. If set to true, the next screens will be shown. If set to false, the next screens will not be shown.

Format

```
void ShowNextScreens(bool bShow)
```

Input Parameters

bool bShow	True or false.
------------	----------------

Return Values

No return values.

ShowProperties

This method shows the properties dialog of the ActiveX control. This dialog can be used to edit the control properties from a graphical user interface.

This method does not return until the user chooses the **OK** or **Cancel** button in the properties' dialog.

Format

```
void ShowProperties()
```

Input Parameters

No input parameters.

Return Values

No return values.

ShowStatusBar

This method shows or hides the status bar.

Format

```
ShowStatusBar(long Show)
```

Input Parameters

long Show	Can either be set to non-zero (show) or zero (hide).
-----------	--

Return Values

No return values.

ShowTitleBar

This method shows or hides the title bar.

Format

```
ShowTitleBar(long Show)
```

Input Parameters

long Show	Can either be set to non-zero (show) or zero (hide).
-----------	--

Return Values

No return values.

ShowToolbar

This method shows or hides the toolbar.

Format

```
ShowToolbar(long Show)
```

Input Parameters

long Show	Can either be set to non-zero (show) or zero (hide).
-----------	--

Return Values

No return values.

Menu Interface Methods

The menu interface methods can be only used when the AllowPopupMenu property is set to "true".

The following methods are available:

- GetMenuHandle
- MenuType
- ShowMenu

GetMenuHandle

Internal only.

MenuType

This method returns the type of the popup menu before it is shown.

Menu types:

Value 1: Main popup

Value 2: Edit popup

Value 3: ListView popup

This method can be used together with the method ShowMenu to show or hide the popup menu, depending of the menu type.

Format

```
short MenuType()
```

Input Parameters

No input parameters.

Return Values

short The menu type.

ShowMenu

This method shows or hides the popup menu, depending on the value passed in the first parameter. If the value is 1, the popup menu will be shown. If the value is 0, the popup menu will not be shown.

Format

```
void ShowMenu(BOOL bShowMenu)
```

Input Parameters

BOOL bShowMenu 1 (show) or 0 (do not show).

Return Values

No return values.

Events

Events are used to notify the container of changes in the state of the control. The following events are available:

Event	Description
void Connected()	This event is fired when the control has successfully been connected to the server. It follows a call to Connect. It is not fired if the connection fails.
void Disconnected()	This event is fired when the control has been disconnected from the server. It can be fired due to a call to Disconnect, or because the server disconnected, or because the user chose the command Disconnect from the menu.
void NewScreenShown()	This event is fired when a new screen is received from the host. It is fired when the dialog is about to appear in the user's screen and not after the new data arrives. At the moment this event is fired, the control has a valid dialog to interact with.
void ScreenSent()	This event is fired after a screen has been sent to the host. When this event has been fired, there is no valid dialog available until NewScreenShown is received again.
void ConnectFailed()	This event is fired when Connect has been called but the connection has failed. A connect can fail if, for example, the server is not accepting connections.
void ShowingPopupMenu(MenuInterface)	<p>This event is fired when the right mouse button is pressed. A pointer to the MenuInterface class is passed in the first parameter, which can be used to get the menu type and to show or hide the popup menu. For example:</p> <pre data-bbox="784 1474 1290 1537">MenuInterface->ShowMenu(TRUE / FALSE) MenuType = MenuInterface->MenuType)</pre> <p>where MenuType is:</p> <ul style="list-style-type: none"> Main popup Menu 0x01 Edit control Menu 0x02 Listview Menu 0x03

Event	Description
void UserDefinedEvent(szText)	This event is fired when the user chooses a push button or static text control for which a user-defined event has been defined in the SDK. szText contains the text defined in the SDK. For further information, see <i>Defining the Control Properties</i> (the description of the Action page) in the documentation <i>Defining the Rules Using the SDK</i> . When a user-defined event is fired, data is not sent to the server (that is: there is no communication with the server).

Sample: User-defined Event Handling

The user-defined event in this sample is used to check and filter the command that has been entered in the dialog. A user-defined event has been defined for the **OK** button (with text "ok-button") and for a static control (with text "other"). The used field IDs depend on the dialog that has been used and can be seen in the SDK.

```
<script language="VBScript">
<!--
Sub cControl_UserDefinedEvent(szText)
if szText="ok-button" Then
    input = cControl.GetFieldContent(3004)
    i = 0
    if input="avs" Then i = i + 1
    if input="myp" Then i = i + 1
    id = 3005
    if i=0 Then call cControl.SetFieldContent(id, "Please enter a valid command")
    if i=1 Then call cControl.SendKey(Return, 3004)
end If

if szText="other" Then
    msgbox "other was chosen"
    end If
end sub
-->
</script>
```